

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

Отчет по лабораторной работе № 2.17
по дисциплине «Основы программной инженерии»

Выполнил студент группы ПИЖ-б-о-22-1

Душин Александр Владимирович.

Подпись студента _____

Работа защищена « » _____ 20__ г.

Проверил Воронкин Р.А. _____
(подпись)

Ставрополь 2024

Тема: Разработка приложений с интерфейсом командной строки (CLI) в Python3.

Цель работы: приобретение построения приложений с интерфейсом командной строки с помощью языка программирования Python версии 3.x.

Ход выполнения работы:


1. Создать общедоступный репозиторий на GitHub с использованием лицензии MIT и язык программирования Python:

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner *

 MrPlatynum ▾

Repository name *

ProgrammEngineering20

⚠ The repository ProgrammEngineering20 already exists on this account.

Great repository names are short and memorable. Need inspiration? How about [scaling-spoon](#) ?

Description (optional)



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Initialize this repository with:



Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: Python ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: MIT License ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set  main as the default branch. Change the default name in your [settings](#).



You are creating a public repository in your personal account.

Create repository

Рисунок 1 – Создание общедоступного репозитория на GitHub с заданными настройками

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents
$ git clone https://github.com/MrPlatynum/ProgrammEngineering20.git
Cloning into 'ProgrammEngineering20'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
```

Рисунок 2 – Клонирование созданного репозитория на локальный компьютер

```
ENV/
env.bak/
venv.bak/

# Spyder project settings
.spyderproject
.spyproject

# Rope project settings
.ropeproject

# mkdocs documentation
/site

# mypy
.mypy_cache/
.dmypy.json
dmypy.json

# Pyre type checker
.pyre/

# pytype static type analyzer
.pytype/

# Cython debug symbols
cython_debug/

# PyCharm
# JetBrains specific template is maintained in a separate JetBrains.gitignore that can
# be found at https://github.com/github/gitignore/blob/main/Global/JetBrains.gitignore
# and can be added to the global gitignore or merged into this file. For a more nuclear
# option (not recommended) you can uncomment the following to ignore the entire idea folder.
.idea/
```

Рисунок 3 – файл .gitignore

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering20 (main)
$ git checkout -b develop
Switched to a new branch 'develop'
```

Рисунок 4 – организация репозитория в соответствии с моделью ветвления git flow

2. Проработайте примеры лабораторной работы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```

```
import argparse
import json
import os.path
from datetime import date
```

```
def add_worker(staff, name, post, year):
    """
    Добавить данные о работнике.
    """
```

```
staff.append({"name": name, "post": post, "year": year})
```

```
return staff
```

```
def display_workers(staff):
```

```
    """
```

```
    Отобразить список работников.
```

```
    """
```

```
    # Проверить, что список работников не пуст.
```

```
    if staff:
```

```
        # Заголовок таблицы.
```

```
        line = "+-{}--{}--{}--{}--".format(
            "-" * 4, "-" * 30, "-" * 20, "-" * 8
        )
```

```
    )
```

```
    print(line)
```

```
    print(
```

```
        "| {:^4} | {:^30} | {:^20} | {:^8} |".format(
            "No", "Ф.И.О.", "Должность", "Год"
        )
```

```
    )
```

```
    )
```

```
    print(line)
```

```
    # Вывести данные о всех сотрудниках.
```

```
    for idx, worker in enumerate(staff, 1):
```

```
        print(
```

```
            "| {:>4} | {:<30} | {:<20} | {:>8} |".format(
```

```
                idx,
```

```
                worker.get("name", ""),
```

```
                worker.get("post", ""),
```

```
                worker.get("year", 0),
```

```
            )
```

```
        )
```

```
        print(line)
```

```
    else:
```

```
        print("Список работников пуст.")
```

```
def select_workers(staff, period):
```

```
    """
```

```
    Выбрать работников с заданным стажем.
```

```
    """
```

```
    # Получить текущую дату.
```

```
    today = date.today()
```

```
    # Сформировать список работников.
```

```
    result = []
```

```
    for employee in staff:
```

```
        if today.year - employee.get("year", today.year) >= period:
```

```
            result.append(employee)
```

```
    # Возвратить список выбранных работников.
```

```
    return result
```

```

def save_workers(file_name, staff):
    """
    Сохранить всех работников в файл JSON.
    """
    # Открыть файл с заданным именем для записи.
    with open(file_name, "w", encoding="utf-8") as fout:
        # Выполнить сериализацию данных в формат JSON.
        # Для поддержки кириллицы установим ensure_ascii=False
        json.dump(staff, fout, ensure_ascii=False, indent=4)

def load_workers(file_name):
    """
    Загрузить всех работников из файла JSON. """
    # Открыть файл с заданным именем для чтения.
    with open(file_name, "r", encoding="utf-8") as fin:
        return json.load(fin)

def main(command_line=None):
    # Создать родительский парсер для определения имени файла.
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "filename", action="store", help="The data file name"
    )
    # Создать основной парсер командной строки.
    parser = argparse.ArgumentParser("workers")
    parser.add_argument(
        "--version", action="version", version="% (prog)s 0.1.0"
    )
    subparsers = parser.add_subparsers(dest="command")
    # Создать субпарсер для добавления работника.
    add = subparsers.add_parser(
        "add", parents=[file_parser], help="Add a new worker"
    )
    add.add_argument(
        "-n", "--name", action="store", required=True, help="The worker's name"
    )
    add.add_argument("-p", "--post", action="store", help="The worker's post")
    add.add_argument(
        "-y",
        "--year",
        action="store",
        type=int,
        required=True,
        help="The year of hiring",
    )
    # Создать субпарсер для отображения всех работников.
    _ = subparsers.add_parser(
        "display", parents=[file_parser], help="Display all workers"
    )

```

```

# Создать субпарсер для выбора работников.
select = subparsers.add_parser(
    "select", parents=[file_parser], help="Select the workers"
)
select.add_argument(
    "-P",
    "--period",
    action="store",
    type=int,
    required=True,
    help="The required period",
)
# Выполнить разбор аргументов командной строки.
args = parser.parse_args(command_line)
# Загрузить всех работников из файла, если файл существует.
is_dirty = False
if os.path.exists(args.filename):
    workers = load_workers(args.filename)
else:
    workers = []
# Добавить работника.
if args.command == "add":
    workers = add_worker(workers, args.name, args.post, args.year)
    is_dirty = True
# Отобразить всех работников.
elif args.command == "display":
    display_workers(workers)
# Выбрать требуемых работников.
elif args.command == "select":
    selected = select_workers(workers, args.period)
    display_workers(selected)
# Сохранить данные в файл, если список работников был изменен.
if is_dirty:
    save_workers(args.filename, workers)

if __name__ == "__main__":
    main()

```

```

C:\Users\Alexander\Desktop\Универ\4 семестр\Основы программной инженерии\ЛР20_ДушинАВ>python ex1.py add workers.json
usage: workers add [-h] -n NAME [-p POST] -y YEAR filename
workers add: error: the following arguments are required: -n/--name, -y/--year

C:\Users\Alexander\Desktop\Универ\4 семестр\Основы программной инженерии\ЛР20_ДушинАВ>python ex1.py add -n Fedor -p Cleaner -y 2005 workers.json

C:\Users\Alexander\Desktop\Универ\4 семестр\Основы программной инженерии\ЛР20_ДушинАВ>python ex1.py add -n Elena -p Cleaner -y 2010 workers.json

C:\Users\Alexander\Desktop\Универ\4 семестр\Основы программной инженерии\ЛР20_ДушинАВ>python ex1.py display workers.json

```

No	Ф.И.О.	Должность	Год
1	Fedor	Cleaner	2005
2	Elena	Cleaner	2010

```

C:\Users\Alexander\Desktop\Универ\4 семестр\Основы программной инженерии\ЛР20_ДушинАВ>python ex1.py select -P 6 workers.json

```

No	Ф.И.О.	Должность	Год
1	Fedor	Cleaner	2005

```

C:\Users\Alexander\Desktop\Универ\4 семестр\Основы программной инженерии\ЛР20_ДушинАВ>

```

Рисунок 5 – Проработанные команды

```

{} workers.json X
C: > Users > Alexander > Desktop > Универ > 4 семестр
1
2 {
3     "name": "Fedor",
4     "post": "Cleaner",
5     "year": 2005
6 },
7 {
8     "name": "Elena",
9     "post": "Cleaner",
10    "year": 2010
11 }
12

```

Рисунок 6 – Файл workers.json

3. Выполнить индивидуальные задания (вариант 7):

Задание 1. Для своего варианта лабораторной работы 2.16 необходимо дополнительно реализовать интерфейс командной строки (CLI).

Листинг кода:

```

"""

```

Для своего варианта лабораторной работы 2.16 необходимо дополнительно реализовать интерфейс командной строки (CLI).

```

"""

```

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

```

```
import argparse
import json
import sys
import os.path
import jsonschema
```

```
def validate_data(data, schema):
    try:
        jsonschema.validate(instance=data, schema=schema)
        return True
    except jsonschema.exceptions.ValidationError as err:
        print(err)
        return False
```

```
def add_train(trains, destination, train_number, departure_time):
    """Добавляет информацию о поезде и сохраняет список поездов в файл."""
    train = {
        'название пункта назначения': destination,
        'номер поезда': train_number,
        'время отправления': departure_time,
    }

    trains.append(train)
    trains.sort(key=lambda x: x['название пункта назначения'])
    return trains
```

```
def list_trains(trains):
    """Выводит список всех поездов."""
    line = f'+-{"-" * 35}-+-{"-" * 15}-+-{"-" * 25}-+'
    print(line)
    print(f'| {'Название пункта назначения':^35} | {'Номер поезда':^15} | {'Время отправления':^25} |")

    for train in trains:
        print(line)
        print(
            f'| {train['название пункта назначения']: ^35} | {train['номер поезда']: ^15} | {train['время отправления']: ^25} |")
        print(line)
```

```
def select_trains(trains, search_time):
    """Выводит поезда, отправляющиеся после указанного времени."""
    found = False
    result = []

    print(f"Поезда, отправляющиеся после {search_time}:")
    for train in trains:
```



```

train_time = train['время отправления']
if train_time >= search_time:
    result.append(train)
    found = True
if found:
    return result

if not found:
    return "Нет поездов, отправляющихся после указанного времени."

def display_help():
    """Выводит справку о доступных командах."""
    print("Список команд:\n")
    print("add - добавить информацию о поезде;")
    print("list - вывести список всех поездов;")
    print("select <время> - вывести поезда, отправляющиеся после указанного времени;")
    print("save <file_name> - сохранить информацию о поездах в файл JSON;")
    print("load <file_name> - загрузить информацию о поездах из файла JSON;")
    print("exit - завершить работу с программой.")

def save_trains(filename, trains):
    """
    Сохранить информацию о поездах в файл JSON.
    """
    with open(filename, 'w', encoding="utf-8") as f:
        json.dump(trains, f, ensure_ascii=False, indent=4)

def load_trains(file_name):
    """
    Загрузить информацию о поездах из файла JSON и выполнить валидацию данных.
    """
    train_schema = {
        "type": "array",
        "items": {
            "type": "object",
            "properties": {
                "название пункта назначения": {"type": "string"},
                "номер поезда": {"type": "string"},
                "время отправления": {"type": "string", "pattern": "^\\d{2}:\\d{2}$"}
            },
            "required": ["название пункта назначения", "номер поезда", "время отправления"]
        }
    }

    try:
        with open(file_name, "r", encoding="utf-8") as f:
            data = json.load(f)
            if validate_data(data, train_schema):
                return data

```

```

        else:
            print("Invalid data format in JSON file.")
    except FileNotFoundError:
        print("File not found")

def main(command_line=None):
    """Основная функция управления программой."""

    # Создать родительский парсер для определения имени файла.
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "filename",
        action="store",
        help="The data file name"
    )
    # Создать основной парсер командной строки.
    parser = argparse.ArgumentParser("trains")
    parser.add_argument(
        "--version", action="version", version="% (prog)s 0.1.0"
    )
    subparsers = parser.add_subparsers(dest="command")
    # Создать субпарсер для добавления поезда.
    add = subparsers.add_parser(
        "add", parents=[file_parser], help="Add a new train"
    )
    add.add_argument(
        "-d", "--destination", action="store", required=True, help="The destination's name"
    )
    add.add_argument("-n", "--train_number", action="store", help="The train's number")
    add.add_argument("-t", "--departure_time", action="store", help="The departure time")

    # Создать субпарсер для отображения всех поездов.
    _ = subparsers.add_parser(
        "display",
        parents=[file_parser],
        help="Display all trains"
    )

    # Создать субпарсер для выбора поезда.
    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Select the train"
    )
    select.add_argument(
        "-t",
        "--departure_time",
        action="store",
        type=str,
        required=True,
        help="The required departure time"
    )

```

```

)

# Выполнить разбор аргументов командной строки.
args = parser.parse_args(command_line)

# Загрузить все поезда из файла, если файл существует.
changed_file = False
if os.path.exists(args.filename):
    trains_list = load_trains(args.filename)
else:
    trains_list = []

# Добавить поезд.
if args.command == "add":
    trains_list = add_train(
        trains_list,
        args.destination,
        args.train_number,
        args.departure_time
    )
    changed_file = True

# Отобразить все поезда.
elif args.command == "display":
    list_trains(trains_list)

# Выбрать требуемый поезд.
elif args.command == "select":
    selected_trains = select_trains(trains_list, args.departure_time)
    list_trains(selected_trains)

else:
    print(f"Неизвестная команда {args.command}", file=sys.stderr)

# Сохранить данные в файл, если список поездов был изменен.
if changed_file:
    save_trains(args.filename, trains_list)

if __name__ == '__main__':
    main()

```

```

C:\Users\Alexander\Desktop\Универ\4 семестр\Основы программной инженерии\ЛР20_ДушинаАВ>python individual1.py add -d Moscow
w -n 245 -t 12:45 trains.json

C:\Users\Alexander\Desktop\Универ\4 семестр\Основы программной инженерии\ЛР20_ДушинаАВ>python individual1.py add -d Stavropol
-n 214 -t 13:30 trains.json

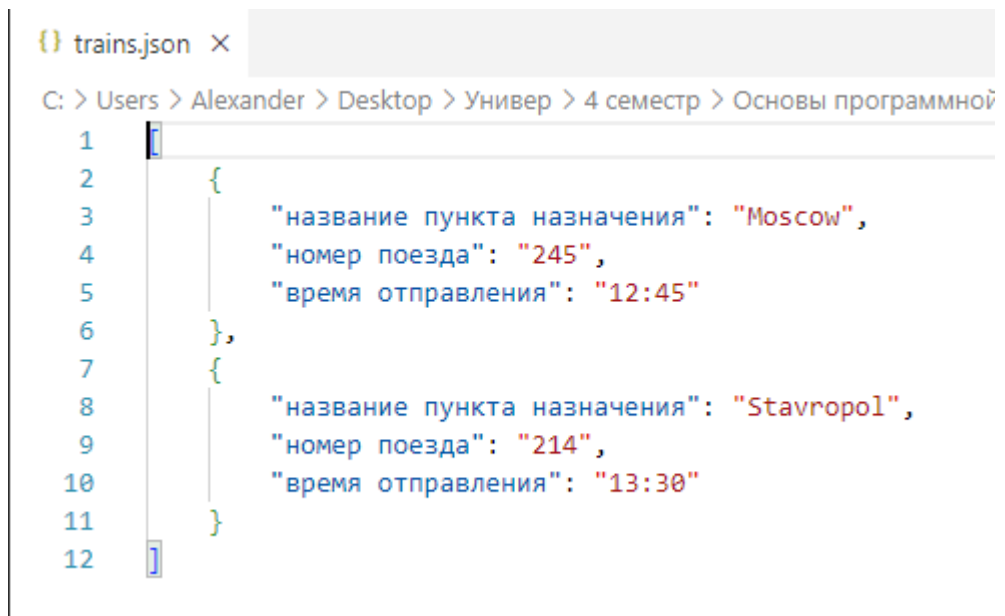
C:\Users\Alexander\Desktop\Универ\4 семестр\Основы программной инженерии\ЛР20_ДушинаАВ>python individual1.py display trains.json
+-----+-----+-----+
| Название пункта назначения | Номер поезда | Время отправления |
+-----+-----+-----+
| Moscow                     | 245          | 12:45              |
+-----+-----+-----+
| Stavropol                   | 214          | 13:30              |
+-----+-----+-----+

C:\Users\Alexander\Desktop\Универ\4 семестр\Основы программной инженерии\ЛР20_ДушинаАВ>python individual1.py select -t 13:00 trains.json
Поезда, отправляющиеся после 13:00:
+-----+-----+-----+
| Название пункта назначения | Номер поезда | Время отправления |
+-----+-----+-----+
| Stavropol                   | 214          | 13:30              |
+-----+-----+-----+

C:\Users\Alexander\Desktop\Универ\4 семестр\Основы программной инженерии\ЛР20_ДушинаАВ>

```

Рисунок 7 – Выполнение команд

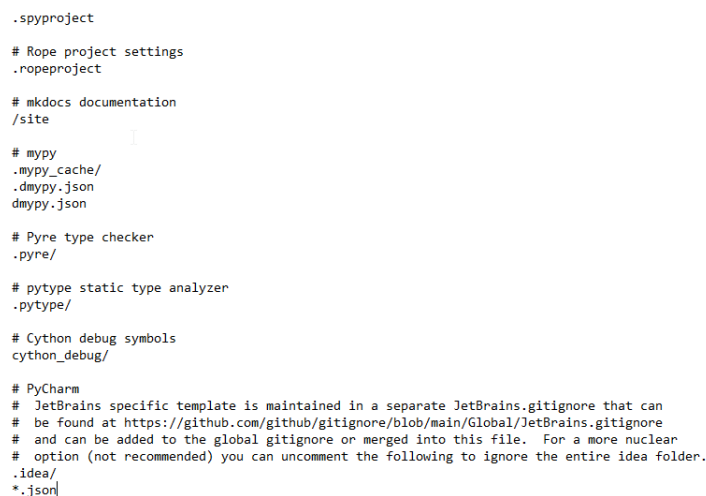


```

1 [
2   {
3     "название пункта назначения": "Moscow",
4     "номер поезда": "245",
5     "время отправления": "12:45"
6   },
7   {
8     "название пункта назначения": "Stavropol",
9     "номер поезда": "214",
10    "время отправления": "13:30"
11  }
12 ]

```

Рисунок 8 – Файл trains.json



```

.spyproject

# Rope project settings
.ropeproject

# makedocs documentation
/site

# mypy
.mypy_cache/
.dmypy.json
dmypy.json

# Pyre type checker
.pyre/

# pytype static type analyzer
.pytype/

# Cython debug symbols
cython_debug/

# PyCharm
# JetBrains specific template is maintained in a separate JetBrains.gitignore that can
# be found at https://github.com/github/gitignore/blob/main/Global/JetBrains.gitignore
# and can be added to the global gitignore or merged into this file. For a more nuclear
# option (not recommended) you can uncomment the following to ignore the entire idea folder.
.idea/
*.json

```

Рисунок 9 – Изменённый файл .gitignore

Задание повышенной сложности. Самостоятельно изучите работу с пакетом `click` для построения интерфейса командной строки (CLI). Для своего варианта лабораторной работы 2.16 необходимо реализовать интерфейс командной строки с использованием пакета `click`.

Листинг кода:

```
"""
Самостоятельно изучите работу с пакетом click для построения интерфейса командной
строки
(CLI). Для своего варианта лабораторной работы 2.16 необходимо реализовать интерфейс
командной строки с использованием пакета click.
"""
```

```
# !/usr/bin/env python3
# -*- coding: utf-8 -*-
```

```
import click
import json
import os.path
import jsonschema
```

```
def validate_data(data, schema):
    try:
        jsonschema.validate(instance=data, schema=schema)
        return True
    except jsonschema.exceptions.ValidationError as err:
        print(err)
        return False
```

```
def add_train(trains, destination, train_number, departure_time):
    """Добавляет информацию о поезде и сохраняет список поездов в файл."""
    train = {
        'название пункта назначения': destination,
        'номер поезда': train_number,
        'время отправления': departure_time,
    }

    trains.append(train)
    trains.sort(key=lambda x: x['название пункта назначения'])
    return trains
```

```
def list_trains(trains):
    """Выводит список всех поездов."""
    line = f'+-{"-" * 35}-+-{"-" * 15}-+-{"-" * 25}-+'
    for train in trains:
```

```

click.echo(line)
click.echo(f"| {'Название пункта назначения':^35} | {'Номер поезда':^15} | {'Время
отправления':^25} |")

for train in trains:
    click.echo(line)
    click.echo(
        f"| {train['название пункта назначения':^35} | {train['номер поезда':^15} | {train['время
отправления':^25} |")
    click.echo(line)

def select_trains(trains, search_time):
    """Выводит поезда, отправляющиеся после указанного времени."""
    found = False
    result = []

    click.echo(f"Поезда, отправляющиеся после {search_time}:")
    for train in trains:
        train_time = train['время отправления']
        if train_time >= search_time:
            result.append(train)
            found = True
    if found:
        return result

    if not found:
        return "Нет поездов, отправляющихся после указанного времени."

@click.group()
def cli():
    """Список команд для управления информацией о поездах."""
    pass

@cli.command()
@click.argument('filename', type=click.Path())
@click.option('-d', '--destination', prompt='Название пункта назначения', help='The destination\'s
name')
@click.option('-n', '--train_number', prompt='Номер поезда', help='The train\'s number')
@click.option('-t', '--departure_time', prompt='Время отправления', help='The departure time')
def add(filename, destination, train_number, departure_time):
    """Добавить информацию о новом поезде."""
    if os.path.exists(filename):
        with open(filename, 'r', encoding="utf-8") as f:
            trains_list = json.load(f)
    else:
        trains_list = []

    trains_list = add_train(trains_list, destination, train_number, departure_time)

```

```
save_trains(filename, trains_list)
```

```
@cli.command()
@click.argument('filename', type=click.Path())
def display(filename):
    """Вывести список всех поездов."""
    if os.path.exists(filename):
        with open(filename, 'r', encoding="utf-8") as f:
            trains_list = json.load(f)
            list_trains(trains_list)
    else:
        click.echo("Файл не найден.")

@cli.command()
@click.argument('filename', type=click.Path())
@click.option('-t', '--departure_time', prompt='Время отправления', help='The departure time')
def select(filename, departure_time):
    """Вывести поезда, отправляющиеся после указанного времени."""
    if os.path.exists(filename):
        with open(filename, 'r', encoding="utf-8") as f:
            trains_list = json.load(f)
            selected_trains = select_trains(trains_list, departure_time)
            if selected_trains:
                list_trains(selected_trains)
            else:
                click.echo("Нет поездов, отправляющихся после указанного времени.")
    else:
        click.echo("Файл не найден.")

def save_trains(filename, trains):
    """
    Сохранить информацию о поездах в файл JSON.
    """
    with open(filename, 'w', encoding="utf-8") as f:
        json.dump(trains, f, ensure_ascii=False, indent=4)

if __name__ == '__main__':
    cli()
```

```

C:\Users\Alexander\Desktop\Универ\4 семестр\Основы программной инженерии\ЛР20_ДушинаВ>python individual12.py add trains.json
Название пункта назначения: Moscow
Номер поезда: 244
Время отправления: 12:30

C:\Users\Alexander\Desktop\Универ\4 семестр\Основы программной инженерии\ЛР20_ДушинаВ>python individual12.py add trains.json
Название пункта назначения: Stavropol
Номер поезда: 545
Время отправления: 14:00

C:\Users\Alexander\Desktop\Универ\4 семестр\Основы программной инженерии\ЛР20_ДушинаВ>python individual11.py select -t 13:00 trains.json
Поезда, отправляющиеся после 13:00:

```

Название пункта назначения	Номер поезда	Время отправления
Stavropol	545	14:00

```

C:\Users\Alexander\Desktop\Универ\4 семестр\Основы программной инженерии\ЛР20_ДушинаВ>python individual12.py display trains.json

```

Название пункта назначения	Номер поезда	Время отправления
Moscow	244	12:30
Stavropol	545	14:00

Рисунок 10 – Выполнение команд

```

{} trains.json X
C: > Users > Alexander > Desktop > Универ > 4 семестр > Основы программн
1  [
2      {
3          "название пункта назначения": "Moscow",
4          "номер поезда": "244",
5          "время отправления": "12:30"
6      },
7      {
8          "название пункта назначения": "Stavropol",
9          "номер поезда": "545",
10         "время отправления": "14:00"
11     }
12 ]

```

Рисунок 11 – Итоговый файл trains.json

4. Зафиксируем сделанные изменения, сольем ветки и отправим на удаленный репозиторий:

```

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering20 (develop)
$ git log --oneline
2990044 (HEAD -> develop) Upload SRC
78278e5 Upload SRC
f612082 (origin/main, origin/HEAD, main) Initial commit

```

Рисунок 12 – Коммиты ветки develop во время выполнения лабораторной работы


```

$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering20 (main)
$ git merge develop
Updating f612082..2990044
Fast-forward
 .gitignore           | 3 +-
 .pre-commit-config.yml | 23 +++++
 environment.yml       | 12 +++
 ex1.py                | 155 +++++++++++++++++++++++++++++++++++++
 flake8.cfg            | 5 ++
 individual1.py         | 202 +++++++++++++++++++++++++++++++++++++
 individual2.py         | 131 +++++++++++++++++++++++++++++++++++++
 pyproject.toml         | 12 +++
 8 files changed, 542 insertions(+), 1 deletion(-)
 create mode 100644 .pre-commit-config.yml
 create mode 100644 environment.yml
 create mode 100644 ex1.py
 create mode 100644 flake8.cfg
 create mode 100644 individual1.py
 create mode 100644 individual2.py
 create mode 100644 pyproject.toml

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering20 (main)
$ |

```

Рисунок 13 – Слияние ветки develop в ветку main

```

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering20 (main)
$ git push origin main
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 12 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (12/12), 6.74 KiB | 3.37 MiB/s, done.
Total 12 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com/MrPlatinum/ProgrammEngineering20.git
 f612082..2990044  main -> main

```

Рисунок 14 – Отправка на удаленный репозиторий

Ответы на контрольные вопросы:

1. В чем отличие терминала и консоли?

Терминал – то программное обеспечение, которое предоставляет доступ к командной строке операционной системы. Он может быть графическим или текстовым. В графическом режиме терминал представляет собой окно, в котором пользователь может вводить команды. В текстовом режиме терминал работает как виртуальный консольный интерфейс, который обеспечивает доступ к командной строке без графического окружения.

Консоль – то окружение, в котором выполняются команды и приложения в операционной системе. Консоль обычно представляет собой текстовое окно или интерфейс, где пользователь может взаимодействовать с операционной системой путем ввода команд. Терминал часто используется для доступа к консоли, но они могут использоваться вместе или отдельно, в зависимости от конкретной среды.

2. Что такое консольное приложение?

Консольное приложение (CLI приложение) – это программа, которая работает в командной строке операционной системы и взаимодействует с пользователем через текстовый интерфейс. Эти приложения обычно используются для выполнения определенных задач, без необходимости графического интерфейса пользователя. Примеры консольных приложений включают утилиты командной строки, административные инструменты и другие инструменты, которые не требуют графического взаимодействия с пользователем.

3. Какие существуют средства языка программирования Python для построения приложений командной строки?

`sys.argv`: Стандартный модуль Python `sys` предоставляет доступ к аргументам командной строки через список `argv`.

`getopt`: Модуль `getopt` используется для разбора аргументов командной строки в стиле UNIX.

`argparse`: Модуль `argparse` предоставляет более мощные и гибкие средства

для обработки аргументов командной строки и создания интерфейсов командной строки в Python.

4. Какие особенности построение CLI с использованием модуля sys?

`sys.argv`: Позволяет получить список аргументов командной строки, переданных при запуске скрипта.

Ограниченный функционал: `sys.argv` предоставляет только базовые средства для работы с аргументами командной строки и не поддерживает определение аргументов с длинными ключами или опций.

5. Какие особенности построение CLI с использованием модуля `getopt`?

Парсинг аргументов в стиле UNIX: `getopt` позволяет разбирать аргументы командной строки в соответствии с соглашениями UNIX.

Поддержка коротких и длинных опций: `getopt` поддерживает как короткие (`-o`) так и длинные (`--option`) опции командной строки.

Более сложное использование: `getopt` требует некоторой дополнительной работы по обработке аргументов и опций, по сравнению с `sys.argv`.

6. Какие особенности построение CLI с использованием модуля `argparse`?

Гибкость и функциональность: `argparse` предоставляет мощные средства для определения аргументов командной строки, их типов, описания и даже автоматической генерации справочной информации.

Поддержка длинных и коротких опций: `argparse` позволяет определять как длинные, так и короткие опции командной строки.

Удобство использования: `argparse` предоставляет простой и интуитивно понятный API для создания интерфейсов командной строки в Python, что делает его предпочтительным выбором для многих разработчиков.